Architecture & Design Decisions

Purpose of this Page

This page captures the rationale behind every non-obvious design choice in the `ssh-baseline` role. Each entry follows the pattern: **What we did → Why → Trade-off accepted**.

Where possible, comments inside the role itself reference these decisions; this page consolidates them in one place.

Identity & Access

AD is the source of truth for SSH public keys

What we did: AD user accounts have their SSH public key stored in the `sshPublicKey` attribute (OpenSSH-LPK schema extension). On Linux, sshd retrieves keys via `AuthorizedKeysCommand /usr/bin/sss_ssh_authorizedkeys %u` (run as `nobody`), which queries SSSD which queries AD.

Why: Centralised key lifecycle — offboarding an AD user revokes their SSH access across every host immediately, without touching each server. Users cannot bypass revocation by maintaining their own `~/.ssh/authorized_keys` because `AuthorizedKeysFile` is globally set to `none`.

Trade-off: AD/SSSD must be available for AD users to log in. The `pbr_admin` break-glass account exists precisely for the case where AD/SSSD is unavailable.

Group membership is the sole gate; no per-user allow lists

What we did: SSSD is configured with `ad_access_filter` restricting login to members of `SG_ServerAccess` or `SG_Sudo`. `realm permit --groups` mirrors the same gate at the realmd layer. sshd's `AllowGroups` enforces it again at the SSH protocol layer.

Why: Three independent layers of group-based access control means a misconfiguration in any one layer cannot accidentally grant broader access. Group changes in AD propagate to every host without any local change.

Trade-off: Defence in depth at the cost of slightly more configuration to keep in sync. The role generates all three from the same variables (`ad_server_access_group`, `ad_sudo_group`), so drift is unlikely.

Break-glass: local pbr_admin account, password auth, source-IP restricted

What we did: The `pbr_admin` local account uses password authentication (only), restricted by sshd `Match` block to source IPs in `10.1.0.0/16,192.168.0.0/16`. It has full sudo with the local password (not AD password, no Duo).

Why: If AD, SSSD, or Duo is unavailable, an administrator can still access every host. Password-only is acceptable here because the account is gated by source-IP and protected by fail2ban.

Trade-off: A local password to manage on each host. Mitigation: the password is in 1Password, rotated on demand, and ssh access is source-IP-restricted to PBR admin networks (default `pbr_admin_allowed_sources`).

Ansible automation account: local user, key-only, NOPASSWD sudo

What we did: The `ansible` account is a local Unix user (not in AD). It authenticates by SSH key only and has `NOPASSWD ALL` in sudoers via `/etc/sudoers.d/ansible`.

Why: Ansible needs deterministic, non-interactive access. Tying it to AD or Duo would block automation during AD/Duo outages and require interactive MFA for every play.

Trade-off: A local account with passwordless sudo is a privileged credential. Mitigations: (1) account password is locked (`passwd -l`) — key authentication only, (2) the public key is unique to the control node, (3) the private key on `pbr-ansible-kl1` is owned by `pbr_admin` mode 0600.

SSH & PAM

AuthenticationMethods publickey,keyboard-interactive

What we did: sshd is configured to require both an SSH publickey *and* a keyboard-interactive PAM challenge. PAM is configured so that Duo is the keyboard-interactive challenge for AD users.

Why: This is Duo's documented Ubuntu integration pattern. PAM rather than `ForceCommand` means the MFA happens at the auth phase before the user's shell starts — including any failure path is logged and rate-limited consistently.

Trade-off: Royal TS's Rebex SSH library cannot do `AuthenticationMethods publickey,keyboard-interactive` directly — it supports one auth method per session. Workaround: set Royal TS authentication method to "Any" in Advanced/Security settings. Native OpenSSH clients (including PowerShell `ssh.exe`) handle it correctly.

AllowGroups includes the local sudo group

What we did: `sshd_config`'s `AllowGroups` directive lists `sudo sg_serveraccess sg_sudo`. The local `sudo` group entry is what permits the local accounts (`ansible`, `pbr_admin`) to log in — they are not AD users and have no AD group membership.

Why: A single `AllowGroups` directive is simpler than multiple `Match User` exceptions. Local accounts qualify via local `sudo`; AD users qualify via either AD group.

Trade-off (and the v2.4.1 fix): Any account that needs SSH access must be in the local `sudo` group. Initially the role assumed the bootstrap had handled this for the `ansible` account, but it had been done manually on the canary and not on later hosts. v2.4.1 added an idempotent task to `preconditions.yml` to enforce it.

AuthorizedKeysFile is globally "none"

What we did: Set `AuthorizedKeysFile none` globally, then re-enable `.ssh/authorized_keys` only inside the `Match User ansible` block.

Why: If `AuthorizedKeysFile` were enabled globally, an AD user could drop their own keys into `~/.ssh/authorized_keys` and bypass the AD-side key revocation that's central to the design. The `ansible` account is local and has no AD-side key, so its `Match` block specifically re-enables local key file lookup.

Trade-off: Slightly non-obvious sshd config. Documented inline in the template.

PAM stack uses pam_succeed_if for break-glass carve-outs

What we did: Both `/etc/pam.d/sshd` and `/etc/pam.d/sudo` use `pam_succeed_if` at the top to detect the break-glass account (`pbr_admin`) and the AD sudo group, branching execution accordingly.

Why: This puts the auth policy in PAM where it can be uniformly logged and audited, rather than depending on multiple sudoers/sshd config layers. It also makes the policy explicit and reviewable in a single file per service.

Trade-off: PAM jump arithmetic (`success=1`, `success=2`, `success=done`) is non-obvious. See the PAM Stack section in the **Duo MFA Integration** page for full explanation.

pam_duo.so referenced by absolute path

What we did: PAM stacks reference `/usr/lib64/security/pam_duo.so` by absolute path rather than relying on PAM's module search path.

Why: Duo's `duo-unix` Debian package installs the module to `/usr/lib64/security/` which is not in Ubuntu's default PAM module search path (Ubuntu expects `/lib/x86_64-linux-gnu/security/`). This is Duo's documented approach for Ubuntu. See https://duo.com/docs/duounix#pam-configuration.

Trade-off: Absolute path is less portable across distributions, but the role only supports Ubuntu so this is acceptable.

Duo MFA

duo-unix from Duo's official APT repo (not Ubuntu universe libpam-duo)

What we did: Install `duo-unix` from Duo's official APT repository (`https://pkg.duosecurity.com/Ubuntu`) and explicitly remove `libpam-duo` / `libduo3` if present.

Why: Inline comment in `roles/ssh-baseline/tasks/duo.yml`:

> 1. Ubuntu universe ships 1.11.3 (2022) which has incompatibilities with current Duo Auth API and returns HTTP 403 in some scenarios.
> 2. Duo's 2.1.0+ is required for the April 2026 CA bundle rotation.
> 3. Duo's docs explicitly target the duo-unix package on Ubuntu 22.04.

Trade-off: An extra APT repository to manage. The role handles GPG key import, repo addition, and legacy package removal automatically.

failmode = safe (not secure)

What we did: `/etc/duo/pam_duo.conf` has `failmode = safe`, meaning if Duo's cloud is unreachable, authentication falls through to single-factor (publickey for SSH, password for sudo).

Why: A Duo cloud outage should not lock administrators out of every Linux host simultaneously. Single-factor publickey is still strong — AD-managed keys with key revocation in effect, plus source-IP restrictions on break-glass.

Trade-off: During a Duo outage, MFA is not enforced. Acceptable because (a) publickey alone is already a strong factor, (b) AD password is still required for sudo, (c) Duo outages are rare and visible.

Duo group restriction limits MFA to AD users

What we did: `pam_duo.conf` has `groups = sg_serveraccess,sg_sudo` (lowercased — SSSD normalises AD group names). pam_duo.so only prompts users in those groups.

Why: Local accounts (`pbr_admin`, `ansible`) should never hit Duo — `pbr_admin` is break-glass (Duo unavailability is exactly when you need it), and `ansible` is automation. The group filter cleanly excludes them.

Trade-off: AD groups must be membered manually. This matches PBR's existing AD-group-driven access management.

sudo timestamp_timeout extended to 30 minutes

What we did: A drop-in at `/etc/sudoers.d/sudo_timestamp_timeout` sets `Defaults timestamp_timeout=30` (default Ubuntu is 15).

Why: Reduces Duo prompt frequency for AD sudo users during typical maintenance sessions. The session-hijack window remains unchanged because the credential cache is per-tty.

Trade-off: Slightly longer interactive sudo grant window. Considered acceptable given the surrounding controls (Duo, AD password, source-IP restriction, fail2ban).

Active Directory / SSSD

ad_gpo_access_control = disabled

What we did: `sssd.conf` sets `ad_gpo_access_control = disabled`.

Why: Per `sssd-ad(5)`, the default is `enforcing`, which evaluates Windows GPO `RemoteInteractiveLogonRight` settings on every SSH login. Any GPO at any parent OU that sets this right (intentionally for Windows servers, or inherited from an ancestor container) would silently deny SSH access to Linux hosts. We use `ad_access_filter` as the sole access control scheme; the sssd-ad(5) manpage explicitly directs disabling GPO control when doing so.

Trade-off: Cannot use Windows GPO to manage Linux SSH access. Acceptable — AD group membership achieves the same control with less surprise.

Explicit DN references in ad_access_filter

What we did: `ad_access_filter` uses full DN references rather than just group names:

```
(|(memberOf=CN=SG_ServerAccess,OU=Security,OU=Groups,DC=pbr,DC=org,DC=au)(memberOf=CN=SG_Sudo,OU=Security,OU=Groups,DC=pbr,DC=org,DC=au))
```

Why: Direct DN references make the filter unambiguous regardless of LDAP search base. If two groups with the same name existed in different OUs, a name-only filter could match the wrong one.

Trade-off: The filter is bound to the current AD structure. If the security groups move OUs, the filter must be updated.

krb5.conf uses DNS SRV discovery (not static KDC list)

What we did: `/etc/krb5.conf` has `dns_lookup_kdc = true` and no static KDC list. SSSD also writes dynamic snippets to `/var/lib/sss/pubconf/krb5.include.d/`.

Why: Resilient to DC topology changes — new DCs are discovered automatically. PBR has 4 DCs across two sites; SRV records let Kerberos route requests appropriately.

Trade-off: DNS must resolve `_kerberos._tcp.pbr.org.au` SRV records correctly. This is the standard AD-integrated DNS pattern, validated during preflight.

PKI

SCEPman as root CA, distributed via the role

What we did: The role downloads the SCEPman root CA from `https://pki.pbr.org.au/ca`, converts DER to PEM, drops it into `/usr/local/share/ca-certificates/pbr-root-ca.crt`, and runs `update-ca-certificates`.

Why: SCEPman is PBR's chosen ADCS replacement. Distributing the root CA via Ansible means every host trusts the internal PKI — including for Palo Alto IPSec tunnels, Proxmox node TLS, AOS-CX switch EST enrollment, and infrastructure-issued certificates.

Trade-off: SCEPman becomes a dependency for the role to complete. Preflight validates the endpoint reachability before the main role runs.

SCEPman /ca quirk: check mode uses uri+GET, real mode uses get_url

What we did: The CA download task is split: in check mode, it validates reachability via `ansible.builtin.uri` with method GET; in real mode it downloads via `ansible.builtin.get_url`.

Why: SCEPman's `/ca` endpoint returns 404 to HEAD requests (ASP.NET Core/Kestrel quirk). `get_url` does a HEAD pre-check in check mode, which would falsely fail.

Trade-off: Slightly more complex task logic. Documented inline in `ca-trust.yml`.

Operational Behaviour

Preflight is a separate role, importable as a standalone playbook

What we did: `roles/preflight/` is independent from `roles/ssh-baseline/`. The `preflight.yml` playbook runs only preflight; `ssh-baseline.yml` runs preflight first, then the baseline. Both playbooks reference `hosts: targets`.

Why: Operators can validate readiness without making changes. The baseline playbook still runs preflight to ensure it never proceeds against an unverified host. Separating the role makes both phases independently testable.

Trade-off: Two roles to maintain. The preflight role is small and changes infrequently.

serial: 1 and any_errors_fatal: true

What we did: Both playbooks run with `serial: 1` (one host at a time) and `any_errors_fatal: true`.

Why: A failed host stops the whole rollout, preventing fleet-wide breakage from a regression. `serial: 1` means at most one host is in a transient state at any time.

Trade-off: Slower rollouts. Acceptable at PBR's scale (currently 5 hosts; expected ceiling ~10-15).

targets group decouples deployment scope from inventory membership

What we did: Inventory has two groups: `linux` (all known Linux hosts) and `targets` (hosts opted-in to baseline deployment). Playbooks use `hosts: targets` exclusively.

Why: Hosts can be in inventory (for fact-gathering, ad-hoc commands, monitoring) without being in the deployment scope. Most importantly, the control node `pbr-ansible-kl1` can be referenced but never targeted by a baseline run.

Trade-off: Two places to add a host. Mitigated by the deployment runbook checklist.

auditd: auto-detect LXC and skip (v2.4.2)

What we did: `manage_auditd: auto` is the default. The role evaluates `ansible_virtualization_type` at runtime: if `lxc`, auditd is skipped. The decision is reported via debug task. `manage_auditd: true` or `false` forces the decision explicitly.

Why: auditd cannot run inside LXC containers — the kernel audit netlink interface is isolated from container namespaces, and AppArmor's `lxc-default-cgns` profile blocks the mount operations auditd needs. Even root in the container cannot bind as primary audit consumer. Forcing auditd would fail with EPERM at the systemd start.

Trade-off: LXC hosts have no local audit log capture. Currently `pbr-graylog-kl1` and `pbr-thingsboard-kl1` are affected. Compliance evidence for those hosts depends on remote logging (Graylog SIEM). Documented in **Known Limitations**.

Bootstrap script lives outside the role

What we did: `scripts/bootstrap-ansible-user.sh` is a 13-line bash script run manually as root on a fresh host, before the host enters Ansible inventory.

Why: Ansible needs a working `ansible` account to run the role; the role establishes that account's *environment* (sudo group membership, etc.) but cannot create the account because there's no way in. The bootstrap solves the chicken-and-egg.

Trade-off: A small manual step. Easier than alternatives like cloud-init or pre-baked images.

no_log on the realm join task (and other secret-handling tasks)

What we did: The `realm join` task in `ad-join.yml` has `no_log: true`. The Duo PAM config task has `no_log: true`. The AD schema check has `no_log: true`.

Why: These tasks handle vault-decrypted secrets (AD service account password, Duo secret key). Logging them would leak credentials into stdout, `tee`'d log files, and CI output.

Trade-off: Failure diagnosis is harder because the actual error message is hidden. Temporary workaround during diagnosis: comment out `no_log`, repro, then restore (with cleanup of tee'd logs).

What We Considered but Didn't Do

retries on realm join (deferred to v2.5)

Three of five hosts deployed needed two attempts to complete realm join, despite proper AD pre-clean. Root cause: AD multi-master replication lag — the join hits a DC that hasn't replicated the deletion of the pre-cleaned computer object. Adding `retries: 2, delay: 30` would mask this transparently. Currently the role remains visible about the behaviour and operators retry manually. To be revisited as a v2.5 enhancement.

Per-VM Windows Server licensing analysis

Out of scope for this role — covered in separate licensing analysis. Mentioned here only because the question came up during baseline rollout planning.

SSH on a non-standard port

Ubuntu 22.10+ and 24.04 LTS use systemd socket activation for OpenSSH by default. Changing `ssh_port` from 22 requires also managing socket overrides under `/etc/systemd/system/ssh.socket.d/`. Avoided complexity for marginal security benefit (port-knocking is security theatre; fail2ban handles the brute-force noise). Documented as a comment in `defaults/main.yml`.

Where to Read Next

- **Deployment Runbook — New Host** — how to execute these design choices in practice
- **PAM Stack Design** (in the Duo MFA Integration page) — the carve-out arithmetic explained line by line
- **Known Limitations, Troubleshooting & Version History** — what we accept, what we plan to address

---

Revision #1
Created 2026-05-13 05:23:15 UTC by PBR_Documentation
Updated 2026-05-13 05:23:15 UTC by PBR_Documentation